SUPERVISED AUTONOMY FOR SPACE TELEROBOTICS

Paul G. Backes
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

Abstract

Supervised autonomy for control of space manipulators is described in this chapter. The supervised autonomy methodology has been successfully utilized for time delayed remote control of unmanned spacecraft. This methodology also provides a safe and effective means for time delayed control of space manipulators. An important feature of supervised autonomy is that commands can be iteratively generated, saved, simulated, and modified before being sent for execution on the remote robotic system. 'l'his provides the necessary command verification for safe execution as well as time delay invariant execution. The features of a supervised autonomy system are explained through the description of an operational laboratory system and example tasks. Evolutionary directions are also discussed.

1 Introduction

Telerobotics continues to provide a means for extending human presence in space as an integral part of the control architecture of unmanned spacecraft. Scientific, data return is maximized within a fixed mission budget by properly separating the spacecraft and instrument control into ground and spacecraft based segments. Since the spacecraft must be reliable and fail-safe, its design is optimized to provide only the required on-board capabilities for communication and control of the spacecraft and instruments to achieve mission success, The Earth based segment of the system generates command sequences which are telemetered to the remote spacecraft. Command generation on the ground based upon updated data from the spacecraft provides the needed system flexibility to achieve mission success. Extensive human and computational resources are available on the ground compared to providing similar resources on the spacecraft. The spacecraft is able to execute command sequences which have been telemetered from Earth as well as to react to anon~a-1OUS situations. Ground based control of remote unmanned spacecraft is an application of supervised autonomous control. The same basic control architecture can be used for other types of space robots such as unmanned rovers [1] and the focus of this Chapter, space manipulators. Further reference to telerobotics will imply control of a manipulator unless otherwise stated.

Manipulators in space can provide valuable extensions of human capability for task execution. A near term application of space telerobotics is on Space Station Freedom. The required astronaut extravehicular activity time required to perform all of the projected maintenance activities on the Space Station is projected to be above the expected available EVA time. Additionally, there will be an extended man-tended phase when the Space Station is manned only part of the time. Telerobotics can be used to perform some maintenance, assembly, and inspection tasks to relieve the astronauts of some duties. Also, through utilization of ground control, many tasks can be performed using telerobotics when there are no astronauts on board the Space Station.

Telerobotics methods can be separated into three types: manual control, supervisory control, and fully automatic control [2, 3, 4]. The distinction between these methods is briefly described here. Sheridan's text [4] provides a good historical perspective and literature review on these approaches to telerobotics. The term teleoperation may be used generically to describe all telerobotics methods but is used here in its more common connotation of manual control. In manual control, all robot motion is specified by continuous input from a human, with no additional motion caused by a computer. In supervisory control, robot motion may be caused by either human inputs or computer generated inputs. In fully automatic control, all robot motion is caused by computer generated inputs.

'I'here are two primary subsets of supervisory control: supervised autonomy and shared control. The distinction between them is the nature of the inputs from the operator. In shared control, operator commands are sent during execution of a motion and are merged with the closed loop motion generated automatically [5, 6, 7]. Therefore, in shared control, all inputs from the operator are not known a priori to execution of a motion since inputs during execution are also used. In supervised autonomy, autonomous commands are generated through human interaction, but sent for autonomous execution [8]. A command can be sent immediately or iteratively saved, simulated, and modified before sending it for execution on the real robot. Also, individual commands can be complete descriptions of the motion [9] or module commands specifying only modifications to the control or monitoring of a specific module of the remote system [10].

'l'he ability to iteratively save, simulate, and modify commands before sending them for execution is a critical feature of supervised autonomy which distinguishes it from other forms of supervisory control. For safety purposes it is valuable to be able to simulate task execution before sending command sequences to the manipulator for task execution. With supervised autonomy, a command or command sequence can be saved, simulated, and modified before sending it for execution on the real robotic system. Safety is achieved by verifying the commands before sending them for execution on the real robots and through real-time monitoring. Commands can be modified and simulated until they are acceptable for execution on the robot. Individual commands can be concatenated into a command sequence which can then be iteratively simulated and modified and inserted into yet a larger sequence, Sequence generation for autonomous spacecraft is a formal process since dangerous or incorrect commands could result in serious damage, loss of unique scientific opportunities (e.g., during a planetary flyby), or loss of the entire spacecraft. In shared control, operator commands are sent immediately to be merged with the autonomous execution, Safety in shared control is achieved either by relying on the operator to input safe motions, or by having real-time autonomous monitoring and modification of the motion specified by the operator.

The term telerobotics implies a separation between the operator and the robot. This separation gives rise to a partitioning of a telerobotics system in to two components: the local-site where the operator resides, and the remote-site where the robots reside [2]. The separation between the operator and robot causes a communication time delay. Time delay is another factor which makes supervised autonomy an important approach for space telerobotics. For Space Station Freedom applications, the projected round-trip communication time delay between an Earth based local-site and Space Station based remote-site is expected to be approximately 7 seconds (mostly in data processing and relay) [11] while round trip time delay to a planetary spacecraft or vehicle is measured in tens of minutes to hours [1]. Teleoperation and shared control become increasingly difficult with time delays due to the continuous real-time inputs [2, 4]. Supervised autonomy overcomes the time delay problem by providing closed loop control at the remote-site based upon autonomous commands generated at the local-site.

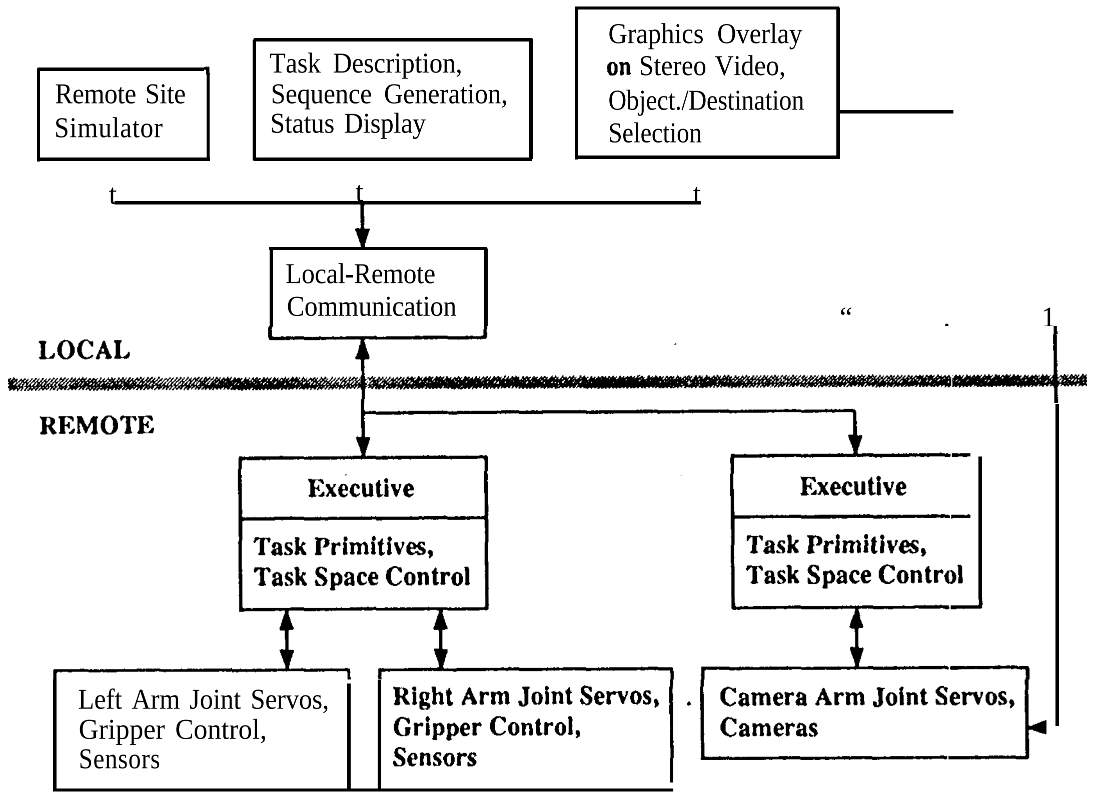
An important feature of supervised autonomy is bounded behavior execution [8]. Bounded behavior execution allows task execution to diverge from the nominally planned motion within a specified bound. For safe operation it is desired to know a priori exactly what the manipulators will do during execution of a task. But, since the remote execution environment cannot be known a priori exactly, real-time execution will rely on both the a priori planned trajectory and perturbations due to remote sensed data. The safety of execution within a specified bound can be tested a priori at the local-site. The remote system can then autonomously monitor execution in real-time to ensure that the state of the motion is within the specified bounds, e.g., deviation from the a priori trajectory or contact force thresholds. If execution moves out the specified bounds, then an automatic reflex action is invoked and further local-site commands are awaited.

The local and remote components of a supervised autonomy system can be divided into subcomponents. The local-site includes sequence generation, sequence analysis, monitoring, and telemetry. The remote-site includes telemetry, command parsing, sequence control, real-time control, monitoring, and reflex. Different implementation approaches may be desired for different application domains. Space applications impose important constraints on the telerobotic system architecture with the flight component on the system usually providing the most stringent constraints. Flight systems require robust flight qualified software running in limited computing environments (limited compared to the ground system). Modification of flight software during flight, although possible, requires an extensive and costly qualification process, This leads to the solution taken for unmanned robotic spacecraft control: command sequences are composed of command types and associated data which specify the desired spacecraft and instrument control behavior [12, 13]. 'l'he flight software is fixed but provides general command types which can be parametrized to generate a wide range of specific control behaviors, This supervised autonomy architecture for programming a remote spacecraft can be used for control of remote space manipulators.

Sequence generation is the process of generating a command sequence which can be telemetered to a remote autonomous robot control system. An operator interface is provided which the operator uses to specify the desired commands. Computer aids can help in the specification of tasks, commands, and parameterization. Computer aids include modeling, visualization, and task planning. Computer modeling provides a model of the manipulated object or task execution environment. Visualization provides a graphical representation of the scene, An accurate representation of the task execution scene is important to ensure that the a priori simulation is a valid representation of the execution that will occur on the real robot. One way to verify that the modeled environment matches the real environment is to overlay a graphical representation of the model on real images of the remote scene, The model can be modified to match the remote scene using data returned from the remote environment [14, 15]. Autonomous task planning aids can provide suggested task commands and parameterization to the operator. Automatic task planning associated with fully automatic control is not yet feasible. To aid the operator, an interface could provide suggested parameterization for specified individual command types [16] and let the operator specify the specific parametrization and sequence of commands. State transition graphs which provide the sequence of commands and automatic parameter selection for a selected task [17] could be provided, but upon failure, reliance on interactive task description is again necessary. Automatic generation of low level command primitives based upon analysis of an operator's actions while interacting with a graphical interface has been suggested [15], but the low level commands do not include context information such as object mass properties and termination conditions which would be provided in a supervised autonomy system.

Sequence analysis determines the expected result of executing a generated sequence and the level of confidence in achieving that result. Graphical simulation is provided so that the powerful analysis capabilities of the human operator can be used. Automatic analysis by the computer may provide tests for dynamic loading, collisions, valid range of motion, and valid commanded velocities and accelerations. It is valuable to have as accurate a model of the remote system as possible to increase confidence in the sequence analysis results. Local-site monitoring analyzes the reports from the remote-site to test for valid execution and system health. The local-site will usually have much greater diagnostics capabilities than the remote-site due to the greater human and computational resources available. Local-site telemetry provides the communication of command sequences to the remote-site and receiving of status and data from the remote-site.

Remote-site telemetry receives command sequences from the local-site and sends status and data to the local-site. Command sequences are parsed at the remote-site into individual commands for execution. Sequence control provides the transitioning of commands. This includes transitioning to the next command in a command sequence upon expected termination and transition to reflex action upon a reflex monitor event. Real-time control provides the closed loop servo control of the remote-site mechanisms. The control is based upon commands generated at the local-site. Remote-site monitoring is the analysis of remote-site execution to provide information on whether to transition the state of execution. Reflex is the ability to respond to monitored conditions. The most common reflex is to transition to the next command in a command sequence based upon a monitor event which indicates that the previous command has successfully completed, An equally impor-



Remote Site Simulator
Task Description, Sequence Generation, Status Display
Graphics Overlay on Stereo Video, Object./Destination Selection
Local-Remote Communication
LOCAL
REMOTE
Executive
Task Primitives, Task Space Control
Executive
Task Primitives, Task Space Control
Left Arm Joint Servos, Gripper Control, Sensors
Right Arm Joint Servos, Gripper Control, Sensors
Camera Arm Joint Servos, Cameras

Figure 1: Laboratory local-remote system block diagram

tant reflex is the ability to transition to a safety reflex action based upon an unexpected monitor event.

'The features and capabilities of a system providing supervised autonomy of a remote manipulator system are described in the following sections through the description of an operational laboratory system.

2 Example Local-Remote System

The local-remote system architecture of the example supervised autonomy system is shown in Figure 1. The same system also provides shared control and force reflecting teleoperation [16, 18, 8], but those capabilities are not within the scope of this Chapter. The primary operator interface workstation (Sun 3/60) provides interactive task description, sequence generation, and status display. A graphics workstation (Silicon Graphics IRIS) provides stereo graphics overlay on stereo video as well as interactive designation of objects or destinations, The remote-site simulator simulates remote-site execution with execution status displayed on the primary workstation and motion displayed on the graphics workstation. 'The remote-site provides two control systems, one for independent, coordinated, or cooperative control of two task execution manipulators, and one for control of a third manipulator for positioning a suite of four cameras. The Executive provides communication with the local-site and initiates task commands as specified by the local-site. Task primitives provide joint and task space control and monitoring of single or dual cooperating manipulators.

2.1 Remote Site System

The remote-site system design of a space telerobotic system has more constraints imposed on it than the local-site system. The resulting capabilities of the remote-site will drive the design of the local-site. A primary remote-site constraint is flight qualification of the software. This creates the need for fixed flight software which has been validated before flight (or modified, validated, and uplinked infrequently). Fixed flight software precludes custom optimized programs for each mission task. Rather, the fixed flight software must provide sufficient functionality to complete both expected and unexpected mission tasks. The solution provided in this laboratory system is a family of parameterizable task primitives, each with general functionality for a class of manipulation tasks. Separate commands provide other needed capability such as databased update, status request, and execution interrupt. Task execution primitives are self contained programs which provide manipulator control capability with behavior as specified by an input parameter set. The control capability is provided via the parameterization but the implementation details are hidden, A natural interface between the local and remote-site systems is then the the parameter lists for the various task primitives,

2.1.1 Executive

The Executive provides functionality similar to that of a spacecraft Command and Data subsystem [19, 12]. It receives commands from the local site, parses the commands to determine command types, and initiates execution of the commands by executing task primitives or other commands with the parametrization given in the command data sets. The Executive also returns system state information to the local-site.

2.1.2 Commands

The interface commands that can be sent to the remote-site by the local-site include database commands, a status command, and execution commands. The Database command has parameters specifying the arm and database datatype followed by the specific database parameters. Remote-site database parameters that can be modified by a database command include a transform specifying the position of the robot's base, force-torque, joint, and singularity safety limit thresholds, grasped object mass properties, and rate to report status to the local-site, The database parameters are used by the task primitives along with the task primitive parameters when executing a task. Database parameters are separate from task primitive parameters because they are expected to change less frequently than task primitive parameters or provide system information. The Status command requests the remote-site to return the state of the arm specified in the command. This is useful when no task is executing and no status information is otherwise being returned. There are six autonomous execution commands (with corresponding remote-site task execution primitives): Cartesian Guarded Motion, Joint Guarded Motion, Move To Touch, Single Arm Generalized Compliant Motion, Dual Arm Generalized Compliant Motion, and Grasp.

Guarded motion task primitives provide free space motion with monitoring for collisions. The Cartesian Guarded Motion primitive [16, 19] performs a single-arm Cartesian interpolated motion and stops on the destination position or sensed force or torque thresholds. Inputs to the primitive include which robot, time or velocity based motion, time or velocity to perform motion in, force and torque thresholds, coordinate frame to sense collision forces in, coordinate frame in which to perform Cartesian interpolation, and position destination via points to go through. The Joint Guarded Motion primitive is the same as the Cartesian Guarded Motion primitive except that joint interpolation is used instead of Cartesian interpolation.

The Move To Touch primitive [16, 19] performs a single-arm move with Cartesian interpolated motion until the specified destination is reached or until a force or torque threshold is exceeded. If a force or torque threshold is reached, then the arm moves back toward its initial position until the force and torque magnitudes are below reverse thresholds, above safety thresholds, or the arm has returned to its initial position. Inputs include which arm, the Cartesian destination, the frame in which to perform Cartesian interpolation, the forward, reverse, and safety force and torque thresholds, and the forward and reverse velocities.

Generalized compliant motion provides general autonomous task execution capability for motion in contact with the environment (as well as shared control). There are both single and dual-arm generalized compliant motion task primitives. The Single Arm Generalized Compliant Motion primitive [9, 6] performs general single-arm Cartesian space compliant motion tasks. Inputs to the primitive include which arm, destination coordinate frame, frame in which to perform Cartesian interpolated motion, frame in which to perform Cartesian force control, frame in which to generate Dither position commands, time or velocity based motion, time or velocity for positional motion, dither magnitude and period, position-force selection vector to select position and force DOFs in the control frame, comply selection vector to select which position DOFs also have compliance, force control gains, gains for stiffness control, force-torque and position-orientation thresholds, a parameter selecting which termination conditions to test for, and termination conditions including maximum errors in position, orientation, force, and torque and their rates of change. The motion in any DOF can have inputs from the position trajectory generator, dither, sensor based control (force, stiffness, etc.), or any simultaneous combination of sources. This approach is similar to impedance control [20] where resultant motion in any DOF is based upon a combination of both the position setpoint and interaction forces, as contrasted with hybrid position-force control [21] where position and force DOFs are separate. Nominal motion generates motion based upon a Cartesian trajectory generator. Sensor based motion (force, stiffness, and dither here) perturbs the nominal motion. Force control modifies the Cartesian setpoint to control contact forces. Stiffness control modifies the Cartesian setpoint to pull the motion back toward the nominal motion, thus counteracting the effects of force control. The Dual-Arm Generalized Compliant Motion primitive provides all of the capabilities of the Single-Arm Generalized Compliant Motion capability and provides cooperative dual-arm control [7].

There are two segments of motion in the Generalized Compliant Motion Primitive, the nominal motion segment and the ending motion segment. When the primitive starts, it executes the nominal motion segment with the specified Cartesian interpolated motion and all other sensors. Motion stops if a monitor event is triggered or Cartesian interpolated motion completes. If the nominal motion segment completes (Cartesian interpolated motion completes), then the end motion segment begins. Exactly the same control occurs except there is no Cartesian interpolated motion; only the sensor based motion is active. But, whereas during the nominal motion segment the termination conditions were not being tested, they are tested during the ending motion and the motion can stop on a monitor event, time, or a termination condition (monitor events and time can also be considered termination conditions).

The Grasp primitive opens or closes the gripper while performing generalized compliant motion. Inputs include the Generalized Compliant Motion inputs as well as finger speed and finger separation. The gripper opens or closes with specified speed and to the specified finger separation and stops upon reaching either a finger threshold force or the specified separation. Generalized compliant motion provides compliance to relieve internal forces during the grasp, or apply specific forces during the grasp, e.g., comply in all DOFs except apply a force against the surface you are grasping.

Dual-arm equivalent Guarded Motion and Move To Touch primitives were not implemented because the chosen evolutionary path for task execution was to fold all capability into a. common modular execution environment. This is discussed below in Section 5.

2.2 Local Site System

The remote-site system design specifies the interface that the local-site can use to control the remote manipulators. The local-site system is then designed to provide the remote-site capability to the operator. There may be multiple local-sites for one remote-site such as for Space Station Freedom where local-sites on Earth and the Space Station could control a common manipulation system [8]. The local-sites could share responsibilities, e.g., the Earth based local-site could generate command sequences and telemeter them to the Space Station based local-site where an astronaut would initiate them. An Earth based local-site is valuable because it has much greater human and computational resources than a space based local-site, This section assumes a single local-site communicating with a single remote-site.

Figure 2: Video-graphics workstation stereo graphics overlay *on* stereo video

2.2.1 Interactive Task Description

Task description and sequence generation are provided by the User Macro Interface (UMI) [16]. UMI abstracts away the details of the local-remote interface and provides the operator with more natural menus for specifying tasks and parametrization. The resulting inputs from the operator are converted to equivalent commands and parametrization to be communicated to the remote-site. An iconic graphical interface implementation of UMI is presently under development, as discussed below in Section 5. The operator has the option of running either the real remote-site robots or simulating the motion at the local-site by sending commands to the remote-site simulator and observing the results on the graphics display. The graphical results are displayed both with wire frame graphics on the primary operator workstation or on the video-graphics workstation. Stereo graphics overlay on stereo video on the video-graphics workstation is shown in figure 2. Simulation mode is selected as a parameter in the UMI environment menu. The remote-site simulator runs identical control software as in the remote-site system and sends joint angle data to the UMI graphics displays.

The operator describes a task by utilizing the interactive UMI menu system. UMI is a hierarchic] menu system which guides the operator from general motion types at the top of the hierarchy to the specific at the bottom of the hierarchy. UMI eventually specifies to the local-site executive task primitives and their parametrization to perform the specific tasks desired by the operator. The operator does not need to know the specific task primitives which will be used, Instead the operator specifies a generic motion type, e.g., guarded motion, move to contact, compliant motion, or grasp and then the interface provides a new macro menu with interaction germane to the specific motion type. For example if the operator specifies compliant motion, the compliant motion menu will appear with hinge, slide, screw, insert, level, push, and general macro options. 'l'he operator will then select one of these and a new menu will appear with inputs pertaining only to that type of motion. The insert menu would allow the operator to specify the insertion direction, force, thresholds, etc. The operator then has only a small number of decisions at any point in the hierarchy but can specify a specific task.

The operator may save a specific parameterization of a task as a task command. For example, the operator can specify a door opening motion but internally it is actually the Generalized Compliant Motion primitive with specific parametrization. 'l'he operator builds the door opening command and can save it by giving it a name, e.g., door_open. The operator can then use the door-open command later either with the simulator or on the real arms. The operator can also string several commands together creating a sequence and save the sequence with a name for later use. Relative and absolute motion commands are useful. A relative motion command has the same relative motion from its starting point as when taught even though the absolute starting point changes. This is useful for tasks executing relative to their environment such as bolt turning and grasping. Absolute motion commands have an absolute position destination independent of where they start their motion. These are useful for moving to an absolute position before beginning a relative motion command.

The sequence editor allows the operator to edit a sequence which has been previously built. The operator selects the sequence menu and then the sequence editor menu and finally the-sequence to edit. The parametrization for each command can then be modified by the operator.

The system Environment is another branch of the tree where the operator specifies parameters which change rarely or are used globally by the different task macros, e.g., which arm (right, left, dual, camera.), time delay to invoke, and safety force-torque thresholds.

The status of the remote-site system is updated on the local-site operator control station monitor whenever a system status (updated at a settable rate, usually approximately 1 Hz) or command result is returned from the remote-site. This includes the graphics simulation which is automatically updated with the remote-site arm positions and the joint and force values which are updated on the graphical displays at the bottom of the interface, as well as the gripper positions.

The stereo graphics overlay on stereo video, shown in Figure 2, is valuable for simulation and interactive destination selection [8]. Tasks are simulated by selecting simulation mode in the environment menu of UMI. Then any command or command sequence that is sent is routed to the remote site simulator for execution. The remote site simulator sends commanded joint angles to the simulator for graphics update. Interactive destination selection is achieved through the use of a spaceball input device which specifies motion of a graphical cursor in the graphics display. The operator selects with an UMI menu to return to UMI the cursor position or the position of the object closest to the cursor. Selecting to return the cursor position allows the operator to interactively generate a path. The operator moves the cursor, presses a button on the spaceball, the motion is simulated, and then the operator can send the same motion for execution on the real robots. ‘This provides a safe interactive means for generating a trajectory through a constrained workspace. The operator can also use the cursor to select an object in the environment. ‘I’he operator moves the cursor near the object and presses a button on the spaceball. The coordinate frame, which is the internal positional representation of the object, closest to the cursor is highlighted (turned red) to indicate the selected object and position. The operator can then graphically simulate a motion to an approach point above the coordinate frame. This command can then be sent to the real robot for execution. Any of these motions can also be saved for later execution or modification.

Graphics overlay on video is used to confirm that the geometrical model of the environment is valid. Operator coached machine vision [14] is used to match the geometric model with the video images returned from the remote environment. With OCMV, the operator provides a rough estimate of an object’s position by manipulating a graphics environment which is overlayed on the returned video. The object’s modeled position is then adjusted for more accuracy with machine vision. Once the model is validated, the video image can be turned off and the graphical scene can be used by itself. This graphical representation of the scene can now be used without the limitations of the fixed viewing characteristics of the real video. The eyepoint can be changed to a more useful point for a given task. This can be very important when the surfaces of interrest are occluded from view. For example, during insertion of an orbital replacement unit on the Space Station, there will likely be no way to see the mating points on the bottom of the ORU with the given camera locations. With a validated graphical representation of the scene, the eyepoint can be moved so that the motion behind the ORU can be seen graphically. This approach to manipulation will change the requirements for cameras in space from what a human would need to accomplish a task to what is necessary to generate a valid geometric model of the environ ment.

3 Sequence Control

Sequence control is control of the transition between commands in a sequence or transition to a reflex action. Two important parts of sequence control are run time binding [8] before execution of a command and testing of termination conditions at the end of a command. Run-time binding binds parameters to a task command just before its execution is initiated. Parameters bound at run time may not be known at the time the command is built. Some examples of run-time binding include binding the current safety parameters, speed factor, and the reporting period to the parameter list. Each command in a task sequence completes due to satisfaction of a termination condition (including safety conditions). If the termination condition is one of the acceptable termination conditions specified in the command, then the next command of the sequence is issued. If not, then a safety reflex action is initiated, and a new command sequence must be sent. Sequences can have interspersed commands for all three remote-site robots and dual-arm cooperative control.

Transition between commands in a sequence can occur at either the local or remote

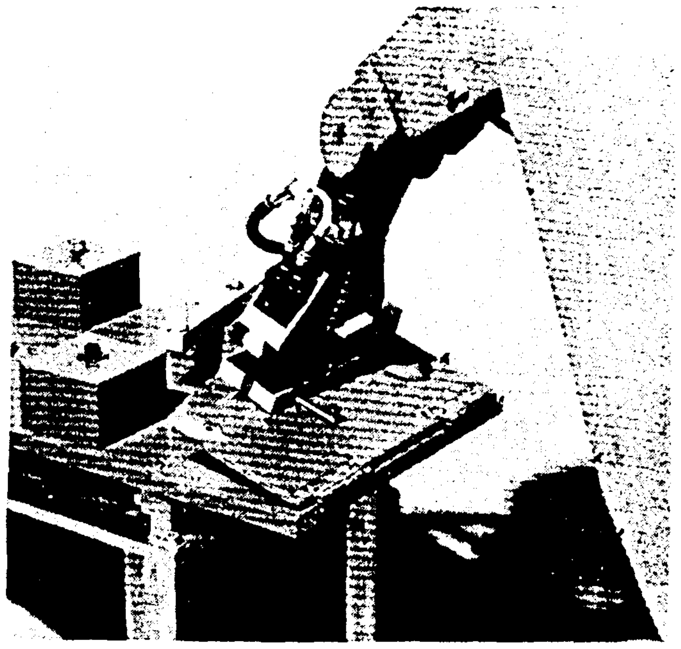


Figure 3: Door opening and closing task

site but transitioning to a reflex action should be done autonomously at the remote site. For sequence control at the local site, a delay at least as long as the round trip time delay will occur between execution of each command in a sequence since the local-site must then receive the remote-site status indicating that the command has terminated successfully before sending the next command in the sequence. Earth based sequence control may be feasible for Space Station or Lunar applications where round trip time delay will likely be less than 10 seconds, but for exploration of the rest of the solar system, the large time delays will make sequence control at the remote site more feasible. Remote site sequence control has been used for unmanned spacecraft missions. In the laboratory system described here, sequence control occurs at the local site with reflex to safety commands done at the remote site,

4 Example Task

Many tasks have been executed using the local-remote system described above [7, 22]. A door opening task sequence is described here to illustrate supervised autonomy for a specific task. The task is to approach, grasp, open, and close a door. The manipulator opening the door is shown in Figure 3. 'The task sequence, *door_task*, is made up of the following commands (with the associated task execution primitive given in parenthesis).

- *gmove_knob_approach* (Cartesian Guarded Move)
- *mtouch_z100mm* (Move To Touch)

- *grasp_close* (Compliant Grasp)
- *door_open_30_deg* (Generalized Compliant Motion)
- *door_close_32_deg* (Generalized Compliant Motion)
- *grasp-open* (Compliant Grasp)
- *gmove_knob_approach* (Cartesian Guarded Move)

The manipulator is initially at a staging location. The *groove-knob-approach* is an absolute motion command to move the manipulator gripper to an approach location above the door. The acceptable termination condition is Stopped On Position Reached. All of the commands except the guarded motion (*gmove_*) commands are relative motion commands meaning they could be used for any location of the door. The guarded motion commands move the arm to the specific location of the door and then away after the task is completed. The *mtouch_z100mm* command moves the gripper into contact with the doorknob using a Cartesian motion of 100 mm along the TOO L frame (att ached to the gripper) Z axis until contact occurs. Actual contact and command termination occurs after approximately 50 mm. The acceptable termination condition is Stopped On Reverse Force which indicates that the motion stopped when moving back toward the starting point and the contact force magnitude fell below the specified reverse force threshold. The *grasp-close* command closes the gripper while applying force control to null any internal forces due to misalignment of the gripper over the knob.

The *door_open_30_deg* and *door_close_32_deg* require the most sophisticated control and monitoring of all the tasks in the sequence so they will be discussed in more detail. In the *door-open-30_deg* command, the Genera.lized Compliant Motion primitive is specified along with specific input parameters. The nominal motion frame, NOM, for Cartesian interpolated motion is specified to have its Z axis along the hinge axis. The force control frame, FORCE, is specified to be on the knob. A Cartesian interpolated motion of 30 degrees about the NOM Z axis is specified. Force control with zero setpoints is specified for all six DOFs of the FORCE frame. Force control is necessary to correct for the difference between the physical motion constrained by the door and the a priori planned nominal motion based upon the model. Stiffness control was specified in all DOFs of the FORCE frame. Stiffness control is necessary to offset the force control based motion when forces are reduced. The acceptable termination condition was specified to be a low orientation error of 0.1 degrees. This is triggered when the actual orientation of the nominal motion frame is within the specified bound during the ending motion, The results are shown in figure 4. The door was successfully opened 30 degrees.

The value of the stiffness control is shown by executing the same task but without use of stiffness control. The results are shown in figure 5. In this case the door opened a maximum of only 21.6 degrees. The maximum rotation occurred when the trajectory generator finished. After that, the ending mot ion time segment began and the door slowly began closing due to its gravity weight. The ending condition of 0.1 degree from the 30 degree goal was never satisfied so the command stopped on the time timeout condition. The reason that the door did not open all of the way is that force control in the FORCE

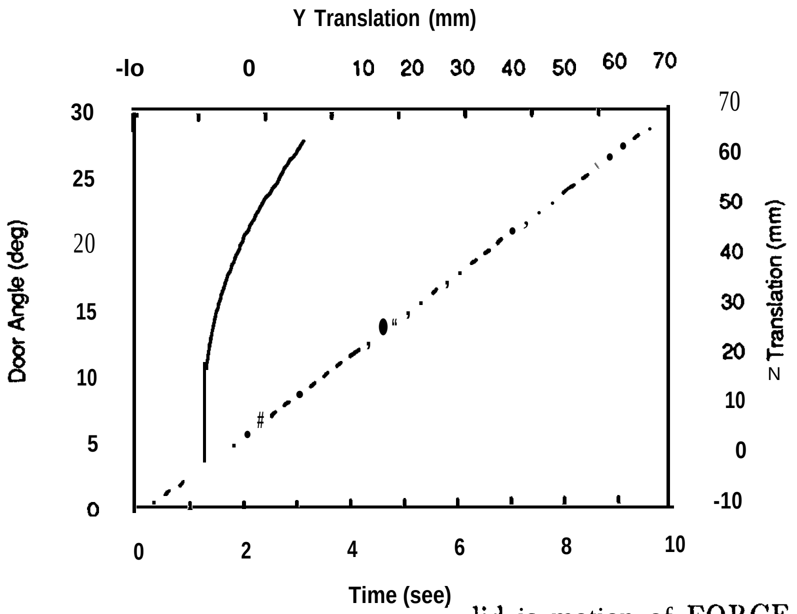


Figure 4: Autonomous door opening results: solid is motion of FORCE frame (knob); dashed is rotation of NOM frame (hinge axis)

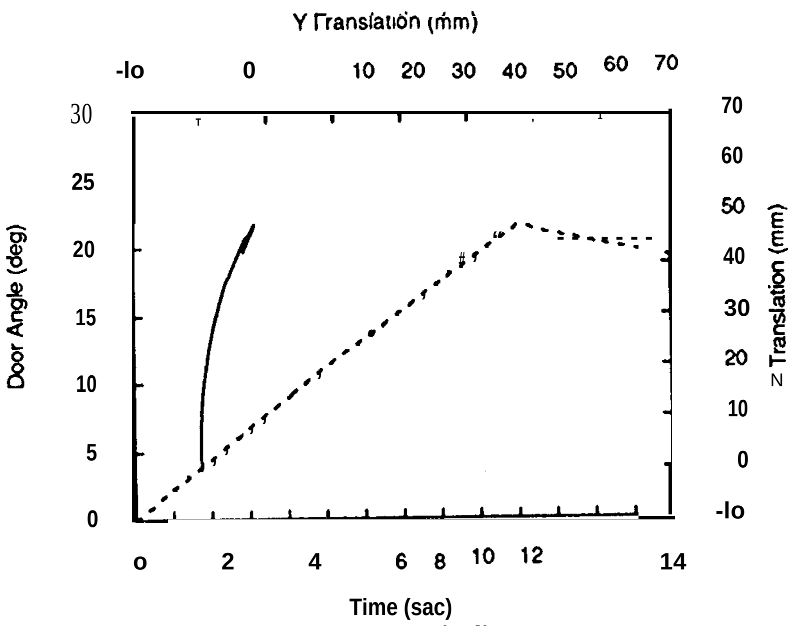


Figure 5: Autonomous door opening results (no stiffness control): solid is motion of FORCE frame (knob); dashed is rotation of NOM frame (hinge axis)

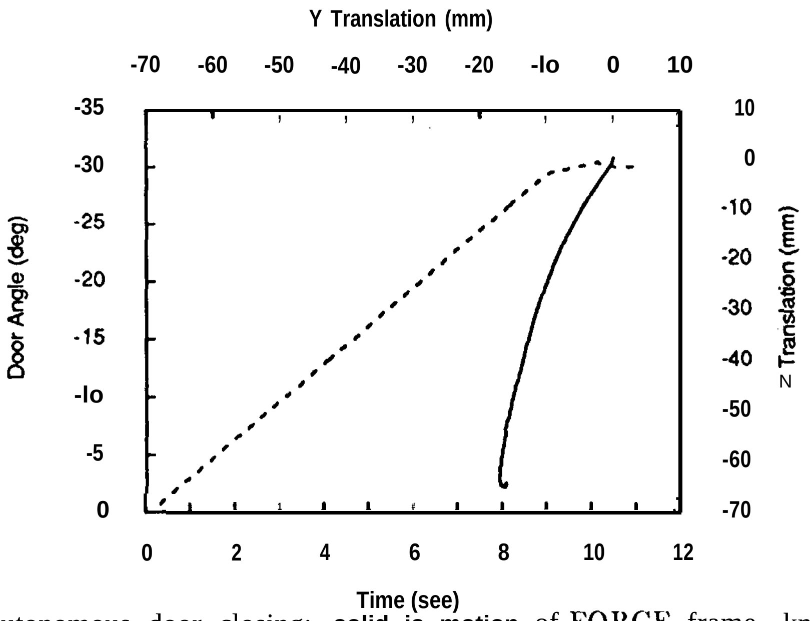


Figure 6: Autonomous door closing: solid is motion of FORCE frame knob ; dashed is rotation of NOM frame (hinge axis)

frame caused motion to resist the nominal trajectory generator motion and there *were* no virtual springs to offset this motion.

The door closing command, *door_close_32_deg*, used the same parameters as for the door opening task, including stiffness control, except that the nominal motion was negative 32 degrees and different termination conditions were used. A 32 degree motion was used to be sure to have the door close completely. If the a priori model was known to be accurate, then a 30 degree specified rotation might have been sufficient, but there will often be a difference between the modeled and physical environments and commands should be robust to this disparity. The specified acceptable termination condition was a combination of low orientation error (3 degrees), low translational error velocity (1 mm/see), and low orientational error velocity (O. 1 degree/see). The translational and orientational error velocities are the rates of change of the translation and orientation errors relative to the nominal motion trajectory. The results are shown in figure 6. The figure shows that the door was successfully closed 30 degrees. The motion is nearly linear until the door makes contact and is closed at 30 degrees. Then the rotation stops which triggers the termination condition,

The rest of the sequence then continues (assuming the termination conditions are always valid) with *grasp_open* to open the gripper, and *gmove_knob_approach*, to move away from the knob.

The door opening and closing commands were shown in more detail to demonstrate the need for the various remote site cent rol feat ures. The door closing task is actually the most difficult because it shows the potential difficulty of determining when a command has completed successfully at a remote task execution site. We could not know exactly how many degrees to close the door. Closed was not a precise distance and orientation state, but rather a distance and orientation state in combination with a motion state constrained by the physical environment. Visual feedback will not be sufficient for many space tasks to determine successful completion. Processed sensory data, such as rate of change of position and orientation errors as used in this example, will be needed to determine successful task completion.

5 Evolution Of The Local-Remote System

'l'he system described above provides the basic capabilities of a supervised autonomy system for space telerobotics. This type of system could be used for near term applications such as telerobotics for maintenance of Space Station Freedom. As telerobotics evolves, more advanced technologies for implementing the capabilities will be provided. A new system is now under development for Space Station Freedom ground-remote telerobotics. Some of the echnologies and implementation approaches for the new system are descr bed below.

5.1 Remote Site Evolution

Three approaches have been considered for increasing the capability of the remote site system: more general task primitives, an interpretive programming language, and data driven execution. The previous evolutionary path of the system described above was to continuously reduce the number of task primitives by increasing the capability of the Generalized Compliant Motion primitive, It would be possible to enhance the Generalized Compliant Motion primitive to include the capability of the two guarded motion primitives and the grasp primitive, thereby further reducing the number of task primitives. This could have two benefits: simplification of the system design and increased capability. The system would be simpler due to the reduced number of primitives, lines of code, and interface specifications. 'J'he system could be more capable since the permutations of the capabilities of the four primitives would be available in one primitive. Including the Move To Touch primitive capability could be more difficult since multiple trajectory segments are included.

An alternative approach to increasing the remote site system capability would be to provide a programming language interface to the local site. This is a common approach for terrestrial robotics [23, 24], This could potentially provide greater capability by the remote site, but flight qualification is likely to be a problem with this approach. The task primitives approach above has the advantage that a primitive's logic, control algorithms, and software can be flight qualified once and then used repeatedly with the only change being the specific parameterization (which of course will have to be verified for safety). Verification of a new program written in a robot control language, before sending it for execution at the remote site, could be difficult and costly. The flexibility gained may not be worth the risk in the approach, particularly for the first generation of space manipulators.

The approach selected to increase the remote site system capability is data driven task execution with fixed software modules [10]. The system design is similar to the remote site architecture for sequence control on the Galileo unmanned spacecraft [12, 13]. The resultant system capability is the permutations of capability of the various control modules (or subsystems in spacecraft control terminology). The implementation for space manipulator control utilizes impedance based Generalized Compliant Motion with extensions for redundant arm control [25, 26]. Any number of control sources, e.g., trajectory generator, force error, and visual servoing, can be used simultaneously. The resultant behavior of the combination of control sources is the task execution. The remote site receives command sequences and stores them in a task command queue or reflex command queues. The task command queue holds the task command sequence to execute the task. The reflex command queues have task sequences which are executed upon monitor events which are not specified as acceptable task termination conditions.

5.2 Local Site Evolution

The local site capability is being upgraded by providing automation aids for model calibration and task planning and an iconic interface for task planning and sequencing integrated with increased graphical interaction for sequence generation. As machine vision capabilities increase, the amount of operator interaction required in operator coached machine vision will decrease. Laser range scanner data will be merged with vision data to further enchance model building and update. Task simulation will be enhanced by adding dynamics and sensor based control effects. For example, arm flexibility and environment contact dynamics will be modeled.

The sequence generation process with the User Macro Interface described above involved creating individual commands from macro templates and then concatenating these commands together into command sequences. An important part of creating individual commands is the generation of context dependent parameterization. This step can be automated if the contextual data is provided to the interface before the command is created. This early step of inputing contextual task parametrization is called the knowledge insertion step and UMI is being enhanced to allow insertion of this data. Thus, context specific commands and parametrization are input into the operator interface during the knowledge insertion step and the operator specifies the sequence of context specific commands for a task during sequence generation, The new system has a knowledgebase (a more general version of the previous database) which holds contextual task execution commands and parametrization, The task specific data may originate from various places such as manufacturer specifications or empirical experimentation. Much of this contextual data would not be known by the operator generating the tasks, but it is automatically provided by the knowledgebase. During sequence generation, the task state must be known so that the contextual information can be automatically generated. Specialists in robotics and task specification can be used for the knowledge insertion phase while a person with different skills, e.g., an astronaut, could perform the sequence generation phase.

Integration of the iconic interface with the graphical environment will allow the operator to better utilize the graphical environment for sequence generation. The previous system provided graphical simulation, object selection. and destination selection. The new system will allow the operator to also select commands within the graphical environment from options automatically generated dependent on the task context. If the context is clear enough, commands could be selected automatically. The teleprogramming methodology was recently proposed where sequences of low level primitive commands are automatically generated by interpreting the actions of an operator interacting with a graphical environment [15]. The approach may be valuable for unmodeled environments, but lacks contextual information and the selection of specialized commands which arc available with a supervised autonomy system. In the supervised autonomy approach, higher level commands are generated, either via the graphical environment or iconic menus. The operator specifies context by selecting objects, destinations, and task types. Commands specific to the task context are generated. These commands are automatically y decomposed into specific remote site task execution commands. For the new remote site task execution system described in Section 5,1 above, the. sequence of module commands and parameterization is automatically generated. If multiple command options are possible, such as grasp compliantly or grasp while pushing against the object, the interface queries the operator for the specific command and any undefined parametrization. This interaction can occur within the graphical environment or on the iconic interface.

The utilization of a priori knowledge insertion and interactive sequence building is now described with a bolt turning example. After inserting an orbital replacement unit into the Space Station truss, a bolt may be need to be tightened to secure the ORU. A " specific torque associated with the ORU will" be required. The specific required torque for that ORU bolt will have been inserted into the interface during the earlier knowledge insertion phase. During sequence generation, the operator selects the high level command to turn the bolt and the operator interface automatically generates a command with detailed parameterization including the required torque read from the knowledgebase based upon the known context of the specific ORU.

6 Conclusions

Space applications provide both an important application domain for telerobotics and many important constraints on the implementation approach. Successful application of supervised autonomy methods to remote control of unmanned spacecraft demonstrates the viability of the approach to that class of space robots. Supervised autonomy is also a viable near term approach for remote control of space manipulators. Safety is achieved by generating commands with parameters specific to the task and through a priori simulation. Effects of time delay are eliminated by providing closed loop control, monitoring, and reflex at the remote site. Remote site computation requirements are limited by providing only task execution and reflex at the remote site while all task planning is done at the local, e.g., Earth, site.

Acknowledgements

The work described in this paper was performed at the Jet Propulsion Laboratory, California institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] Brian H. Wilcox. Robotic vehicles for planetary exploration. *Journal of Applied Intelligence*, 2:181-193, 1992.

[2] W. R. Ferrell and T. 13. Sheridan. Supervisory control of remote manipulation. *IEEE Spectrum*, pages 81-88, October 1967.

[3] Thurston I,. Brooks III. and Thomas 11. Sheridan. Superman: A system for supervisory manipulation and the study of human/computer interactions. Technical Report MITSG 79-20, Massachusetts institute of Technology, July 1979.

[4] Thomas Sheridan. *Telerobotics, A utomation, and Human Supervisory Control.* M.I.T. Press, 1992.

[5] S. Hayati and S. T. Venkataraman. Design and implementation of a robot control' system with traded and shared control capability. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 1310-1315, 1989.

[6] Paul G. Backes. Generalized compliant motion with *sensor* fusion. In *Proceedings 1991 ICA R: Fifth International Conference on A dvanced Robot its, Robots in Unstructured Environments*, pages *1281-1286,* Piss, Italy, June 19-221991.

[7] Paul G. Backes. Dual-arm supervisory and shared control space servicing task experiments. In *Proceedings A IAA Space Programs and Technologies Conference,* Huntsville, AL, March 24-271992. AIAA paper No. 92-1677.

[8] Paul G. Backes. Ground-remote control for space station telerobotics with time delay. In *Proceedings A AS Guidance and Control Conference,* Keystone, CO, February 8-12 1992. AAS paper No. 92-052.

[9] Paul G. Backes. Generalized compliant motion task description and execution within a complete telerobotic system. In *Proceedings I EEE International Conference on Systems Engineering,* August 9-11 1990.

[10] Paul G. Backes, Mark K. Long, and Robert). Steele. Designing minimal space telerobotics systems for maximum performance. In *Proceedings A IA A Aerospace Design Conference*, Irvine, CA, February 3-61992.

[11] R. Aster, J.M. de Pitahaya, and G. Deshpande. Analysis of end-to-end information system latency for space station freedom. Jet Propulsion Laboratory, Internal Document D-86.50, May 1991.

[12] Galileo Project. Galileo program description document - command and data subsystem, phase 9.1. '1'ethnical Report 625-355-06000, D-535 Rev. G, Jet Propulsion Laboratory, May 1989.

[13] Galileo Project. Galileo flight operations plan - galileo command dictionary. Technical Report 1'11625-505, D-234, Jet Propulsion Laboratory, September 1989.

[14] B. Don, B. Wilcox, T. Litwin, and I). Gennery. Operator-coached machine vision for space telerobotics. In *SPIE Symposium on Advances* in *Intelligent Systems, Conference on* Cooperative *intelligent Robots in Space,* Boston, Massachusetts, November 1990.

[15] Janez Funda, Thomas S. Lindsay, and Richard P. Paul. Teleprogramming: Toward delay-invariant remote manipulation. Presence, 1(1):29-44, Winter 1992.

[]6] Paul G. Backes and Kam S. Tso. Umi: An interactive supervisory and shared control system for telerobotics. In *Proceedings IEEE International Conference on Robotics and Automation,* pages 1096--1101, Cincinnati, Ohio, May 1990.

[17] J. Balaram and H. Stone. *Intelligent Robotic Systems For Space Exploration,* chapter Automated Assembly in The JPI, Telerobot Testbed. Kluwer Academic Publishers, 1992.

[18] Samad Hayati, Thomas Lee, Kam Tso, and Paul G. Ba.ekes. A testbed for a unified teleoperated-autonomous dual-arm robotic system, In *Proceedings IEEE International Conference on Robotics and Automation,* 1990.

[19] Kam S. Tso, Paul G. Backes, Thomas S. Lee, and Samad Hayati. A multi-arm tele/autonomous executive system. In *Proceedings international Symposium on Robotics and Manufacturing,* Burnaby, D. C., Canada, July 18-201990.

[20] N. Hogan. Impedance control: An approach to manipulation: Part i – theory. *ASME Journal of Dynamic Systems, Measurement, and Control,* 107:1-7, March 1985.

[21] M. 11. Raibert and J. J. Craig. Hybrid position/force control of manipulators. *ASME Journal of Dynamic Systems, Measurement, and Control,* 102:126-133, June 1981.

[22] Paul G. Backes. Supervised autonomous control, shared control, and teleoperation for space servicing. In *Proceedings Space Operat ions, A pplicat ions, and Research Symposium,* Houston, August 4-6 1992.

[23] UNIMATION Inc. User's guide to val ii, programming manual. Technical Report 398AGI, Shelter Rock Lane, Danbury, CT. 06810, .

[24] L.R. Nackman, M.A. Lavin, R .H. Taylor, W.C. Dietrich, and D.D. Grossman. Aml/x: A programming language for design and manufacturing, In *Proceedings Joint Computer Conference,* pages 145-1.59, November 2-61986.

[25] Paul G. Backes. Multi-sensor based impedance control for task execution. In *Proceedings IEEE International Conference on Robotics and Automation,* Nice, France, May 1992.

[26] Mark K. Long and Paul G. Backes. Impedance based shared control of a redundant robot. In *IASTED International Conference on Control ond Robotics,* pages 106–109, Vancouver, Canada, August 4-61992.